

AD-A052 735

CHARLES STARK DRAPER LAB INC CAMBRIDGE MA
JOVIAL STRUCTURED DESIGN DIAGRAMMER (JSDD). VOLUME IV. USER'S M--ETC(U)
FEB 78 G GODDARD, M WHITWORTH, E STROVINK F30602-76-C-0408
R-1120-VOL-4 RADC-TR-78-9-VOL-4 NL

UNCLASSIFIED

| OF |
AD
A052 735



AD A 052735

RADC-TR-78-9, Vol IV (of four)
Final Technical Report
February 1978

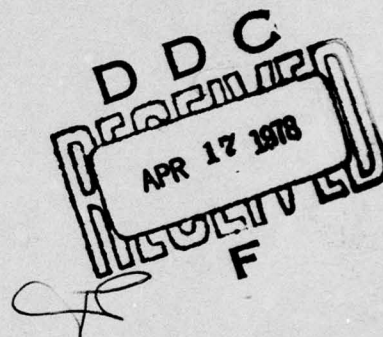
②



JOVIAL STRUCTURED DESIGN DIAGRAMMER (JSDD). *Volume 4*
User's Manual.

G. Goddard
M. Whitworth
E. Strovink

The Charles Stark Draper Laboratory, Inc.



ADJ NU. FILE COPY
DDC

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

18 RADCL

19 TR-78-9-VOL-4

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
RADC-TR-78-9, Vol IV (of four)		
4. TITLE (and Subtitle)	5. TYPE OF REPORT & PERIOD COVERED	6. PERFORMING ORG. REPORT NUMBER
JOVIAL STRUCTURED DESIGN DIAGRAMMER (JSDD). User's Manual Volume IV, User's Manual.	Final Technical Report - September 76 - October 77,	R-1120-VOL-4
7. AUTHOR(s)	8. CONTRACT OR GRANT NUMBER(s)	
G. Goddard, M. Whitworth E. Strovink	F38602-76-C-0408	
9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
The Charles Stark Draper Laboratory, Inc. 555 Technology Square Cambridge MA 02139	P.E. 62702F J.O. 55811412	
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE	
Rome Air Development Center (ISIM) Griffiss AFB NY 13441	February 1978	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	13. NUMBER OF PAGES	15. SECURITY CLASS. (of this report)
Same	31	UNCLASSIFIED
16. DISTRIBUTION STATEMENT (of this Report)	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
Approved for public release; distribution unlimited.	N/A	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
Same		
18. SUPPLEMENTARY NOTES		
RADC Project Engineer: Donald VanAlstine (ISIM)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Structured Programming Structured Design Diagram Structured Extension Parse Parser Generator	Preprocessor Flowcharter JOVIAL J3 Invocation Diagram	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
The report presents the user's view of the JOVIAL Structured Design Diagrammer along with user options and other information about running the programs.		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408 386 JOB

USER'S MANUAL

This document was produced to satisfy the requirements of contract number F30602-76-C-0408 with the Rome Air Development Center. It is one of four companion volumes:

* JOVIAL Structured Design Diagrammer (JSDD)
Report Summary

This document is a summary of the contents of the JSDD Final Report.

* JOVIAL Structured Design Diagrammer (JSDD)
Final Report

This volume presents the design techniques for implementing the JSDD and describes the use of Structured Design Diagrams.

* JOVIAL Structured Design Diagrammer (JSDD)
Program Description

This volume presents a detailed description of the program implementation for purposes of maintaining and/or modifying the JSDD.

* JOVIAL Structured Design Diagrammer (JSDD)
User's Manual

This volume presents the user's view of the JSDD along with user options and other information about running the program.

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUS: LOCATION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	SPECIAL
A	

Acknowledgement

This report was prepared by The Charles Stark Draper Laboratory, Inc., under Contract F30602-76-C-0408 with the Rome Air Development Center at Griffis Air Force Base.

Especial credit is due Margaret Hamilton, who pioneered principles of Structured Programming at Draper Laboratory. Saydean Zeldin originally suggested the symbology implemented in the output of the JOVIAL Structured Design Diagrammer. Thanks should go also to William Daly, who created the Structured Design Diagrammer for the HAL language (currently being used on the NASA Space Shuttle project). The authors are indebted to Victor Voydock for his invaluable assistance in implementing a complete MULTICS user interface which was used successfully for the duration of the JSDD implementation. The authors are also grateful to J. Barton Dewolf whose many suggestions were of great assistance throughout this effort.

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
1. Introduction _____	5
2. Structured Design Diagram Description _____	7
3. Invocation Diagram Description _____	13
4. Running the JSDD System _____	16
5. Detailed Options _____	19
6. Control Cards for Running the JSDD _____	25
Appendix A. The DDG and Invocation Diagrammer Options Compool	

1. Introduction

This document is a user-oriented description of the input/output files and operating procedures necessary to run the JOVIAL Structured Design Diagrammer (JSDD).

The JOVIAL Structured Design Diagrammer is designed to run on all Honeywell Information Systems Inc. Series 6000 computers providing for application program use at least one disk drive, one line printer, and 96K of user memory. Additionally, the computer system should include as much extra memory and as many extra input/output devices as are required to execute the JSDD under control of the Honeywell Information Systems Inc. Series 60 Level 66 and Series 6000 General Comprehensive Operating Supervisor (GCOS) Version 1/G. The consequences of running the JSDD on another computer or operating system have not been determined. The fact that the JSDD is written in JOVIAL J3 (a JOCIT compiler) raises hopes that it may in fact be portable.

The JSDD produces a graphical representation of the control and processing structure of programs written in the JOVIAL J3 programming language, with or without structured extensions (see Final Report, Section 6). It can be thought of as the first component of an integrated software analysis and documentation system which addresses itself to the problem of standardizing the loose collection of software design guidelines known as "structured programming." This first component consists of an automated documentation system which produces two types of diagrams: Structured Design Diagrams (SDDs) and Invocation Diagrams. SDDs provide a graphic display of program control logic. Invocation Diagrams are a display of a software system's functional (calling) structure.

The experienced systems programmer will find the SDD and Invocation Diagram valuable aids in understanding unfamiliar programs. The SDD makes nested control logic transparent and readable, while the Invocation Diagram provides a detailed control map at a procedural level of abstraction. Both of these tasks must be completed manually in the absence of automated tools.

The JSDD has additional utility in system design applications, because by its graphical representation it highlights use of unstructured programming constructs as

well as poorly-considered control paths. Thus, it can be used as a tool to encourage (and with incorporation into a larger software analysis and documentation system, enforce) good programming practices.

The use of the JSDD programs requires an understanding of the JSDD diagrams produced as output; thus, the User's Manual is arranged to reflect this prerequisite. First, Section 2 discusses SDDs and Section 3 discusses Invocation Diagrams. Section 4 then gives a general overview of JSDD running procedures. Section 5 expands upon Section 4 by describing the various options available to the JSDD user. Finally, Section 6 introduces the control cards necessary to run the JSDD programs in the MULTICS GCOS Encapsulator environment.

2. Structured Design Diagram (SDD) Description

Structured Design Diagrams (SDDs) provide a graphic two dimensional display of the nested logical sequences that define the structure of a computer program. SDDs for JOVIAL J3 are constructed from the two basic structural elements shown in figure 2-1.

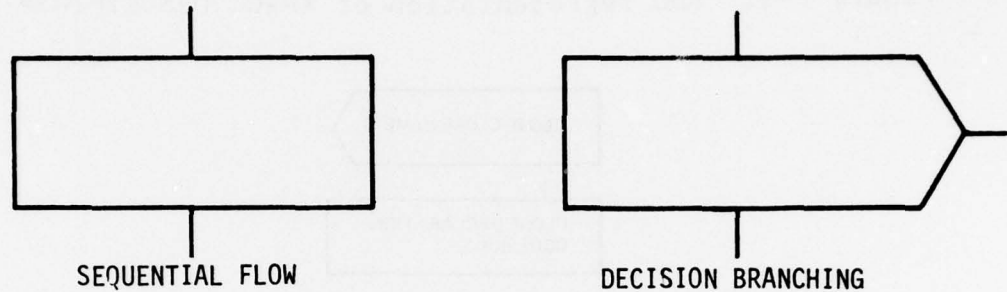


Figure 2-1. SDD Primitives

The rectangular box is used to contain elements that are executed in sequence. Control enters from the top or left side of the rectangle. Each element in a rectangle is executed in sequence and control flows through the bottom of the box.

The pentagonal box is used to contain two types of JOVIAL constructs: module heads and decision making elements. A module head is a <PROGRAM HEAD>, <PROC DESCRIPTOR> or <CLOSE HEAD> (see the syntax definition of JOVIAL in Section 5 of the JSDD Final Report). Control passes through the bottom of a module head's pentagonal box to the module's code body. Figures 2-2, 2-3 and 2-4 illustrate SDD representations of module heads.

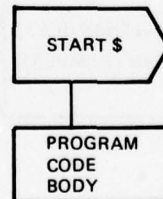


Figure 2-2. SDD representation of <PROGRAM HEAD>

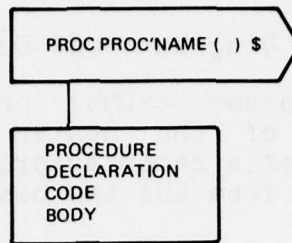


Figure 2-3. SDD representation of <PROC DESCRIPTOR>

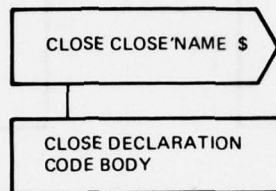


Figure 2-4. SDD representation of <CLOSE HEAD>

A decision making element is a JOVIAL construct which directs the flow of control to one of two paths. Evaluation of the contents of the pentagonal box determines the path to which control is passed. Figures 2-5, 2-6, 2-7 and 2-8 illustrate the SDD representations of JOVIAL's decision making elements. Non-standard decision making elements have been introduced as structured extensions to JOVIAL J3. The structured extensions are the Do While Loop, the Do Until Loop and the Case Statement. Full descriptions of these new constructs are available in Section 6 of the JSDD Final Report.

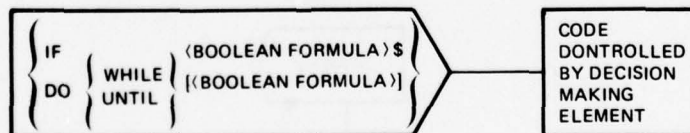


Figure 2-5. SDD representation of the If Statement, Do while Loop and Do Until Loop

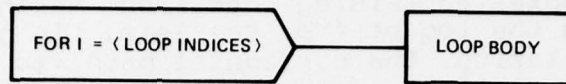


Figure 2-6. SDD representation of the For Loop

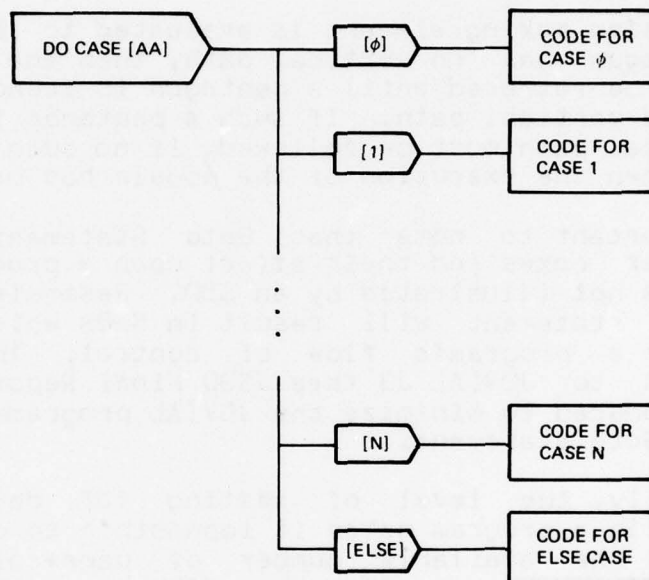


Figure 2-7. SDD representation of the Do Case Statement

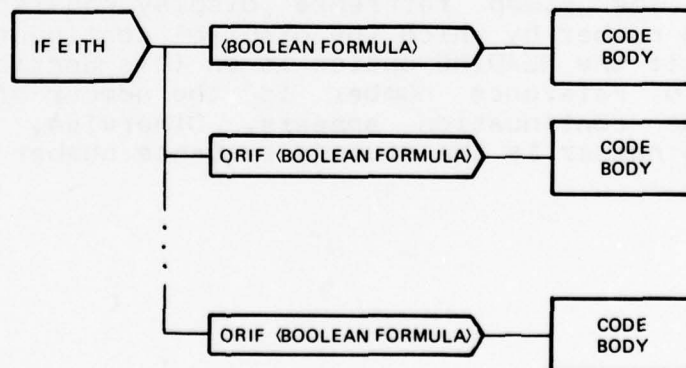


Figure 2-8. SDD representation of the Alternative Statement

Pentagonal boxes containing decision making elements are entered from the top or from the left. In general, they are exited by taking the horizontal path (to the right) or by taking the vertical path (through the bottom of the pentagon). The horizontal path is taken if the decision element is evaluated to be TRUE. Otherwise, the vertical path is taken. Note that the SDD representations of the Do Case Statement and Alternative Statement contain "DO CASE" and "IFEITH" decision making elements. These elements are always evaluated to be TRUE.

If a decision making element is evaluated to be false and its pentagon has no vertical path, then the SDD execution path must be retraced until a pentagon is found which has an unexecuted vertical path. If such a pentagon is found, then the vertical path must be followed. If no such pentagon is found, then the execution of the module has been completed.

It is important to note that Goto Statements appear in rectangular boxes and their effect upon a program's flow of control is not illustrated by an SDD. Restrained usage of the Goto statement will result in SDDs which will better illustrate a program's flow of control. The structured extensions to JOVIAL J3 (see JSDD Final Report, Section 6) were introduced to minimize the JOVIAL programmer's reliance upon the Goto Statement.

Occasionally, the level of nesting (of decision making elements) in a program makes it impossible to display a code block in the available number of page columns. In such cases, it is necessary for the JSDD to create what is referred to as a stump. A stump is a diagram continuation. When the width of a page is such that the display of a code block can not be accommodated, that code block's logical position in the SDD is filled with a stump reference display. The stump reference display consists of a stump reference number by which the diagram continuation can be located. If the HEADING option is on (see Section 5.2), then the stump reference number is the number of the page on which the continuation appears. Otherwise, the stump reference number is the stump's sequence number.

The JSDD recognizes three types of comments: in-line comments, type-1 (or same line) comments and type-2 (or C-type) comments. In-line comments are comments which are embedded in a JOVIAL statement. In-line comments are displayed in their embedding statements in SDDs. A type-1 comment is a comment which begins on the same line as a JOVIAL statement in the input file. In SDDs, type-1 comments appear below the statements to which they refer. A type-2 comment is a comment which appears by itself in the input file. SDDs display type-2 comments next to the line which connects the code blocks which precede and succeed the comment.

Figure 2-9 is a sample page from a design diagram produced by the JSDD system.

BEST AVAILABLE COPY

**C S DRAPER LABORATORY JOVIAL STRUCTURED DESIGN DIAGRAMMER
DESIGN DIAGRAM OF TRANSFER-WRITE3**

[illegible]

Figure 2-9. Sample page of SDD output

3. Invocation Diagram Description

The Invocation Diagrammer produces two different outputs: (1) a list of procedures that are members of one or more recursive invocation loops, and (2) the Invocation Diagram itself.

The first output, if it appears on the diagram, occurs before the actual diagram under the heading "ULTIMATELY SELF-RECURSIVE." Under it are listed all procedures that call themselves, either directly or indirectly. An example of a direct recursive call is a procedure which contains, as part of its code, a call to itself. Indirectly recursive calls are best illustrated, again, by an example. Suppose procedure A can call procedure B which can call procedure C. If, as part of its code, procedure C contains a call to procedure A, all three procedures (A, B, and C) can theoretically call themselves.

The Invocation Diagram supplies recursion information for two reasons. First, the diagrammer has to detect recursive procedures because its algorithm for producing the Invocation Diagram is itself recursive. A recursive procedure could thus cause the diagrammer to diagram forever. Therefore, recursive procedures are only expanded once in the diagram, and thereafter are simply printed and flagged. Since this affects the readability of the diagram, the recursion information ought to be summarized for the user.

Secondly, it can be argued that recursion has no place in JOVIAL programs (JOVIAL J3 does not support recursion), and thus should be banned. The recursion information can be thought of as a warning to the programmer that illegal recursion is a possibility in the submitted program structure. Note that the Invocation Diagrammer only reports the possibility of recursion - it is quite possible that the program in question avoids actually making a recursive call.

The second output is the Invocation Diagram itself. The diagram comes in two parts: a main procedure diagram and diagrams of continuations and independent routines. The diagram of the main procedure, if it occurs, occurs first. Invocation Diagrams are quite simple to read - all procedure names which are connected horizontally to a vertical line are called by the procedure whose name started the vertical line. If the main program exists, it is the top level of

the diagram. For example:

```
/
/--PROC1
/ /
/ /--PROC2+
/ /
/ /--PROC3
/ / /
/ / /--PROC4
/ /
/ /--PROC5*
```

In this example, PROC1 calls PROC2, PROC3, and PROC5. Additionally, we see that PROC3 calls PROC4, and some invisible procedure calls PROC1. That procedure is at the top level of this particular diagram, because PROC1 hangs off the leftmost vertical line possible. There is still more information here; we know PROC2 is an external procedure because it is flagged with a "+". PROC5 is recursive - it is flagged with a "*".

Main diagram continuations (caused by running off the right side of the page) and independent procedures (procedures not called by any of the procedures which are directly or indirectly called by the main program) occur at the end of the diagram, under the heading "CONTINUATIONS AND INDEPENDENT ROUTINES." Continuations consist of "stumps" which correspond to similar "stumps" in the main diagram. If confronted by:

```
/
/-----3
```

in the main program, the user can find the continuation below, which starts with:

```
-----3
/
/
etc.
```

Stump continuations occur in numerical order, after independent procedure diagrams.

The aim of the diagrammer is to diagram all procedures,

regardless of whether they are called (directly or indirectly) by the main program. However, the fact that a procedure is not called directly or indirectly by the main program is not a guarantee that it will appear as an "independent" procedure. It is quite possible that a second "independent" procedure whose diagram is output before that of the first procedure may call the first. In that case, the first procedure's "independent" diagram is suppressed. Nevertheless, the first procedure's diagram will have been generated as part of the second procedure's diagram. No procedure will ever remain both uncalled and undiagrammed.

4. Running the JSDD System

The JSDD system consists of three programs: the Design Diagram Data Base Generator (DDDG), the Design Diagram Generator (DDG) and the Invocation Diagrammer.

The DDDG performs a syntactic analysis of the input JOVIAL program and outputs a three file data base (for use by the DDG and the Invocation Diagrammer). The DDDG is designed to analyze any program that has been compiled by JOVIAL J3 with no error or warning messages. Only one JOVIAL J3 constraint has been altered - the largest DEFINE directive that may occur is one containing 132 characters (instead of 300). However, neither of these limits is a real constraint on DEFINE directive size, since DEFINES can be nested to any depth.

The DDG outputs a Structured Design Diagram (SDD) of the input program in the format specified by the preset variables in the compool OPT (see Section 5.2).

The Invocation Diagrammer outputs an invocation diagram of the input program in the format specified by the preset variables in the compool OPT.

Since all diagram formatting is performed by the DDG and Invocation Diagrammer, one execution of the DDDG on an input program is sufficient to produce diagrams having a wide variety of formats.

The DDDG (DDDG.OBJ) must be loaded with the object segments SYNTH.OBJ, NTABLES.OBJ, DATA.OBJ and SPOOL.OBJ. The DDDG requires five files:

(logical unit number)

- 11 The JOVIAL program to be diagrammed.
- 12 The DDDG message file. This file will contain any error messages generated by the DDDG execution.
- 13 FILE 1 (see the JSDD Program Description Section 4.4). This file is part of the DDG data base.
- 14 FILE 2 (see the JSDD Program Description, Section 4.4). This file is part of the DDG data base.
- 15

FILE 0 (see the JSDD Program Description, Section 4.4). This file is the data base used by the Invocation Diagrammer.

The DDG (DDG.OBJ) must be loaded with the object segments OPT.OBJ, DEBUG.OBJ and SPOOL.OBJ. The DDG operates on ten files:

(logical unit number)

11

FILE 1 (from the DDDG).

12

The DDG message file. This file contains error messages and debugging messages generated by a DDG execution.

13

FILE 2 (from the DDDG).

14

FILE 3'1 (FILE 3 version 1 - see the JSDD Program Description, Section 4.5.1 and Appendix C). This file is part of the DDG intermediate data base and can be destroyed after DDG execution.

15

FILE 4'1 (FILE 4 version 1 - see JSDD Program Description Section 4.5.1 and Appendix C). This file is part of the DDG intermediate data base and can be destroyed after DDG execution.

16

PUTOUT'1 (temporary diagram version 1). This file can be destroyed after DDG execution.

17

FINAL'OUT. This file contains the SDD produced by the DDG.

18

FILE 3'2 (FILE 3 version 2). This file is part of the DDG intermediate data base and can be destroyed after DDG execution.

19

FILE 4'2 (FILE 4 version 2). This file is part of the DDG intermediate data base and can be destroyed after DDG execution.

20

PUTOUT'2 (temporary diagram version 2). This file can be destroyed after DDG execution.

The Invocation Diagrammer (INVOC.OBJ) must be loaded with the object segments OPT.OBJ and SPOOL.OBJ. The Invocation Diagrammer operates on three files:

(logical unit number)

- 11
FILE 0 (from the DDDG).
- 12
The Invocation Diagrammer message file. This file will contain any error messages generated by the Invocation Diagrammer.
- 13
The invocation diagram.

Section 6 contains the control cards necessary to run the JSDD components on the MULTICS GCOS simulator.

5. Detailed Options

This section lists and describes detailed options for running each of the JSDD programs. Sections 5.1, 5.2, and 5.3 discuss options for the Design Diagram Database Generator, the Design Diagram Generator, and the Invocation Diagrammer, respectively.

5.1 DDDG Options

Although most formatting of the design diagrams should be done with the DDG options, there are two options available with the DDDG which directly impact the content and appearance of the final SDDs.

These options are set by means of comment "toggles", which are inserted into the source input file by the user either as extra comments or as additional text in existing comments. The special form of the toggle allows it to be distinguished from ordinary comment text. Toggle syntax is as follows:

- (1) `""<any text> [<toggle name>] <any text>""`
or:
- (2) `""<any text> ['<toggle name>] <any text>""`

Only the first toggle in a comment is processed; all others are ignored.

The syntax in case (1) above means that the toggle is to be turned on; case (2) means that it is to be turned off. These actions are forced regardless of the toggle's current state.

For example,

```
""COMMENT TEXT COMMENT TEXT[EXPAND]COMMENT TEXT""
```

would turn on the EXPAND toggle, while

```
""COMMENT TEXT COMMENT TEXT['EXPAND]COMMENT TEXT""
```

would turn it off.

The two toggles currently available are EXPAND and ASIS. When EXPAND is on, the expanded text of each DEFINE directive (macro) name is substituted for the name. EXPAND toggles can appear anywhere in the source input.

The ASIS toggle is more complex, but its use can lead to much improved SDDs. When ASIS is on, each complete source input text line becomes one line in the output SDD. The utility of this can easily be seen; suppose that a user had set up a portion of a program (say, data declarations) in a special manner, so that certain items appeared under certain columns. He/she would not want the Design Diagrammer to chop up this carefully constructed pattern, as it almost certainly would. The solution is to insert ASIS "brackets" around the code in question. The Design Diagrammer will then preserve this special code "as-is".

The only rules governing placement of ASIS toggles are:

- 1) ASIS toggles must come in pairs, or "brackets"; for every instance of an ASIS activation, there must be a de-activation.
- 2) ASIS brackets can only occur where it would be semantically and syntactically legal to place JOVIAL BEGIN-END brackets (this does not include use of BEGIN and END in array or table declarations).

Failure to follow these rules will invariably result in serious DDDG errors and fatal DDG errors.

There is another toggle implemented, called DEBUG. This causes a history of the DDDG parse to be written to the error file. Although useful for debugging, and for those interested in the mechanics of an LALR(k) parse, DEBUG is of no value to the ordinary user.

5.2 DDG Options

The DDG accepts options which permit the user to specify a wide variety of JOVIAL Structured Design Diagram formats.

The options are defined in a common block in the OPT compool (see Appendix A).

There is no facility currently available for setting the DDG options. Alteration of options involves editing the options compool and recompiling it.

The following is a list of option declarations and descriptions of their meanings and uses.

ITEM DISPLAY'DELIM B P O \$

If DISPLAY'DELIM is on (i.e. preset to 1), BLOCK DELIMITERS are displayed on the design diagram. Otherwise, they are not.

BLOCK DELIMITERS are:

- 1) BEGINS which start COMPOUND STATEMENTS
- 2) ENDS which terminate COMPOUND STATEMENTS and ALTERNATIVE STATEMENTS
- 3) [END DO]
- 4) [END CASE]

DISPLAY'DELIM also controls the printing of COMMENTS and LABELS associated with the BLOCK DELIMITERS.

It is recommended that DISPLAY'DELIM be turned off unless the input program has important COMMENTS or LABELS associated with BLOCK DELIMITERS.

ITEM DOUBLE'SPACE B P O \$

If DOUBLE'SPACE is on, program text within the diagram is double spaced, otherwise, text is single spaced.

Double spacing can increase the execution time of the DDG significantly if the input program is large (because of the double buffering system—see Section 4.5.3 of the JSDD Program Description).

It is recommended that diagrams of large programs be single spaced.

ITEM MARGIN I 36 S P 5 \$

MARGIN sets the left margin of the diagram. The above declaration will cause five blank columns to begin each line of the diagram.

ITEM MESS'SW I 36 S P O \$

MESS'SW directs error and debug messages (see JSDD Program Structure Section 7.2) to either a terminal or to an output file. If the value of MESS'SW is preset to 0, then output is directed to the file whose device number is 12. Otherwise, output is directed to the user terminal.

ITEM PAGE'LNTH I 36 S P 60 \$

PAGE'LNTH should be set to the number of print lines on the paper on which the diagram is to be printed. PAGE'LNTH cannot be used to reserve space at the bottom of pages for page footers. A footer feature is not available on this version of the JSDD.

ITEM PAGE'WIDTH I 36 S P 132 \$

PAGE'WIDTH is the number of the rightmost column which is to be used for printing. The readability of diagrams is improved as the difference of PAGE'WIDTH and MARGIN increases (the number of stumps decreases). Execution time can be improved by setting PAGE'WIDTH to a high value and MARGIN to a low one because fewer output lines are required.

ITEM ST'MAX S P 35 \$

ST'MAX is the maximum number of columns which may be spanned by the text of a statement unit before wraparound occurs.

In general, a low valued ST'MAX will produce a diagram having fewer stumps than a high valued ST'MAX. However, a diagram produced with a low valued ST'MAX may be difficult to read.

The value of PAGE'WIDTH should be considered when assigning a value to ST'MAX.

ITEM HEADING B P 1 \$

If HEADING is on page headings are displayed at the top of each diagram page. Also, pages are numbered, stumps are referenced by page numbers and a table of contents is available.

If HEADING is off, no page headings are displayed, pages are not numbered, stumps are referenced by sequence number and no table of contents is available.

The HEADER, PGM'NAME, NAME'INDEX and HEAD'NO options described below relate to the HEADING option.

ARRAY HEADER 10 h 150 \$

The HEADER array contains the text to be displayed as page headings (if HEADING is on). Elements of HEADER must be preset. For example:

```
BEGIN
21H(DRAPER LAB DIAGRAMMER)
11H(DIAGRAM OF )
END
```

In this case, page three's heading might appear:

DRAPER LAB DIAGRAMMER PAGE 3
DIAGRAM OF PROGRAM'NAME

HEADER (\$O\$) will always be followed by the page number which will start in column PAGE'WIDTH-10. Care should be taken to avoid overwriting the last ten columns.

ITEM PGM'NAME h 150 \$

PGM'NAME contains the name of the program being diagrammed. The contents of PGM'NAME will appear in the page headings of pages displaying the diagram of the main program.

ITEM NAME'INDEX I 36 S P 1 \$

NAME'INDEX is the index into HEADER of the text line in which the name of the program, procedure or close being diagrammed will appear. In the above example, NAME'INDEX was set to one. NAME'INDEX may be set to a negative number if display of the module name is not desired.

ITEM HEAD'NO I 36 S P 1 \$

HEAD'NO is the index of the last element of HEADER which was preset. In the above example, HEAD'NO would have been set to one.

Normally the JSDD leaves one blank line between the page heading and the diagram text. The number of intervening blank lines can be increased by presetting additional HEADER elements to IH() and incrementing HEAD'NO accordingly.

ITEM TABLE'OF'CONTENTS B P 1 \$

If TABLE'OF'CONTENTS and HEADING are both on, then a table of contents is generated for the diagram. The table lists the modules displayed in the diagram and the pages on which the displays begin.

The table of contents appears after the title page (if there is a title page).

ITEM TITLE'SW B P 1 \$

TITLE'SW controls the printing of the title page. If it is on, the title page appears on page one (and subsequent pages, if more are necessary). The array TITLE and TITLE'NO also relate to the TITLE'SW flag.

ARRAY TITLE 70 H 150 \$

The TITLE array contains the text to be displayed on the title page. Its elements are preset the same way that HEADER's elements are preset.

ITEM TITLE'NO I 36 S P I \$

TITLE'NO is the index into TITLE of the last preset element.

5.3 Invocation Diagrammer Options

The Invocation Diagrammer has only three user options, PAGE'WIDTH, PAGE'LENGTH, and PGM'NAME. All of these options are set exactly the same way as they are for the DDG. In fact, the Invocation Diagrammer uses an exact copy of the DDG options compool for its own options. This being the case, the same compool object segment can be loaded for the Invocation Diagrammer that is loaded for the DDG.

Thus, in most cases it will not be necessary to set Invocation Diagrammer options - simply load the options compool object segment used to run the DDG. However, if the DDG was not run, or a PAGE'WIDTH, PAGE'LENGTH, or PGM'NAME (title name) change needs to be made, refer to Section 5.2 for detailed instructions and default attributes.

6. Control Cards For Running the JSDD

The following sections give detailed GCOS Encapsulator control cards for running the various programs which make up the JOVIAL Structured Design Diagrammer (JSDD). These control cards are applicable to the GCOS Encapsulator system available under the MULTICS operating system. A mapping from these control cards to actual GCOS control cards is straightforward, made necessary only by the differences between MULTICS and GCOS file system structure.

Cards common to all decks are SNUMB, IDENT, LOWLOAD, OPTION, LIMITS, and ENDJOB. All these are standard, except for LOWLOAD and LIMITS. LOWLOAD is present because of bugs in both JOVIAL and the GCOS Encapsulator loader. The JOVIAL J3 compiler supplied for this contract generates incorrect function linkages which overwrite areas of low core at execution time. The LOWLOAD card (\$ LOWLOAD 10000) causes a blank area of 10000 words to precede all programs. This ensures that nothing is loaded into the region that JOVIAL may overwrite. As for the GCOS Encapsulator loader, it simply does not function properly without the LOWLOAD card.

The LIMITS card is included in all control card decks. 84k is the minimum core region that should be specified (92K for the DDG) - the time limit field should vary with the size of the program to be diagrammed. Time limits in the sample control cards are extremely high; the built-in JOVIAL default should be adequate in most cases.

There is no execute card in any of the sample control card decks - this is because that card is included in the "canned" deck which is inserted at run time in place of:

```
$          select      >misc_libraries>jocit>execute
```

This "canned" deck consists of the following cards:

```
$          library z*,*z
$          execute dump
$          prmf1     z*,r,r,>ml>jocit>jovlib.020377
$          prmf1     *z,r,s,>ml>jocit>oldlib.020377
```

The dump option can be deleted at the user's discretion.

Sections 6.1, 6.2, and 6.3 contain and describe the control cards used to execute the DDDG, the DDG, and the Invocation Diagrammer, respectively.

6.1 DDDG Control Cards

The control cards needed to execute the DDDG are:

```
$      snumb      efs
$      ident      Strovink.5581c1412
$      lowload    10000
$      option     jovial
$      select     ph18.obj
$      select     ntables.obj
$      select     data.obj
$      select     synth.obj
$      select     spool.obj
$      select     >misc_libraries>jocit>execute
$      limits     90,84k
$      prmf1      10,w,s,jovwrk>ph18.mon.list
$      prmf1      11,w,s,ph24.gcos
$      prmf1      12,w,s,error24
$      prmf1      13,w,s,ph24f1
$      prmf1      14,w,s,ph24f2
$      prmf1      15,w,s,ph24f0
$      endjob
```

Object files loaded by this deck are: ph18.obj (DDD), ntables.obj (parsing tables), data.obj (global variable declarations), synth.obj (external SYNTH procedure), and spool.obj (compool object file for string package).

Files used by DDDG are: 10 (not currently used), 11 (source input to DDDG), 12 (error file), 13 (DDD output file 1 ("FILE 1")), 14 (DDD output file 2 ("FILE 2")), and 15 (DDD output file 0 ("FILE 0")).

6.2 DDG Control Cards

The control cards needed to execute the DDG are:

```
$      snumb      mhw
$      ident      Whitworth.5581c1412
$      lowload    10000
$      option     jovial
$      select     ph24.obj
$      select     spool.obj
$      select     opt.obj
$      select     debug.obj
$      select     >misc_libraries>jocit>execute
$      limits     999,128k
$      prmf1      10,w,s,jovwrk>ph24.mon.list
```

```

$      prmfl      11,w,s,ph24f1
$      prmfl      12,w,s,blah
$      prmfl      13,w,s,ph24f2
$      prmfl      14,w,s,ph24f31
$      prmfl      15,w,s,ph24f41
$      prmfl      16,w,s,tmp1
$      prmfl      17,w,s,out'ph24f
$      prmfl      18,w,s,ph24f32
$      prmfl      19,w,s,ph24f42
$      prmfl      20,w,s,tmp2
$      endjob

```

Object files loaded by this deck are: ph24.obj (DDG), spool.obj (compool object file for string package), opt.obj (option compool object file), and debug.obj (debug toggle compool object file).

Files used by the DDG are: 10 (not currently used), 11 ("FILE 1" from DDDG), 12 (error file), 13 ("FILE 2" from DDDG), 14 (first of double-buffered pair of temporary files referred to as "FILE 3"), 15 (first of double-buffered pair of temporary files referred to as "FILE 4"), 16 (first of double-buffered pair of temporary output files), 17 (permanent output file), 18 (second of double-buffered pair of temporary files referred to as "FILE 3"), 19 (second of double-buffered pair of temporary files referred to as "FILE 4"), 20 (second of double-buffered pair of temporary output files).

Files 14,15,16,18,19, and 20 can and should be deleted following execution of the DDG. This can be accomplished in the control card deck by appropriate changes in the file cards. Files 11, 12, and 13 should be deleted with discretion, since another Design Diagram with different format specifications may be desired.

6.3 Invocation Diagrammer Control Cards

The control cards needed to execute the Invocation Diagrammer are:

```

$      snumb      efs
$      ident      Strovink.5581c1412
$      lowload    10000
$      option     jovial
$      select     invoc.obj
$      select     spool.obj
$      select     opt.obj
$      select     >misc_libraries>jocit>execute
$      limits     25,84k

```

```
      prmfl      10,w,s,jovwrk>invoc.mon.list
$      prmfl      11,w,s,tstf0
$      prmfl      12,w,s,errors
$      prmfl      13,w,s,ph24
$      endjob
```

Object files loaded by this deck are: invoc.obj (Invocation Diagrammer), spool.obj (compool object file for string package), and opt.obj (option compool object file).

Files used by the Invocation Diagrammer are: 10 (not currently used), 11 ("FILE 0" from DDDG), 12 (error file), 13 (Invocation Diagrammer output).

REFERENCES

1. Dahl, O.-J., Dijkstra, E. W., and Hoare, C. A. R., Structured Programming, Academic Press, New York, 1972.
2. Hamilton, M., and Zeldin, S., Top-Down, Bottom-Up Structured Programming and Program Structuring, (Revision 1), Charles Stark Draper Laboratory, Inc., Cambridge, Ma. - E-2728, December 1972.
3. McGowan, C. L., and Kelly, J. R., Top-Down Structured Programming Techniques, Petrocelli/Charter, New York, 1975.
4. Standard Computer Programming Language for Air Force Command and Control Systems, Short Title: CED 2400, Air Force Manual AFM 100-24, Reprint dated 21 April 1972.
5. Structured Programming Series, Volume I, Programming Languages Standards, Final Report, IBM Corporation, FSD 74-0288, March 15, 1975.
6. DeRemer, F. L., Practical Translators for LR(k) Languages, Ph.D. Thesis at the Massachusetts Institute of Technology, Cambridge, Massachusetts, Sept. 1969.
7. Lalonde, W. R., An Efficient LALR Parser Generator, Technical Report CSRG-2, M.Sc. Thesis, University of Toronto, Toronto, Ontario, 1970.
8. McKeeman, W. M., Horning, J. J., and Wortman, D. B., A Compiler Generator Implemented for the IBM System/360, Prentice Hall, 1970.

Appendix A. The DDG and Invocation Diagrammer Options Compool

All DDG user options are contained in the compool OPT which is shown below. Setting options involves changing the preset values in OPT and recompiling the result.

```
start $
  " This is the options compool for instructions  "
  " in setting options , see JSDD User's Manual.  "
common options $
begin
  item display'delim b p 0 $
  item double'space b p 0 $
  item margin i 36 s p 5 $
  item mess'sw i 36 s p 1 $
  item page'length i 36 s p 60 $
  item page'width i 36 s p 132 $
  item st'max i 36 s p 30 $
  item heading b p 1 $
  array header 10 h 150 $
begin
  57h(c s draper laboratory jovial structured design diagrammer)
  18h(DESIGN DIAGRAM OF )
end
  item pgm'name h 150 p 21h(the design diagrammer) $
  item low'lim i 36 s p 20 $
  item max'width i 36 s p 40 $
  item name'index i 36 s p 1 $
  item head'no i 36 s p 1 $
  item table'of'contents b p 1 $
  item title'sw b p 1 $
  array title 70 h 150 $
begin
  1h( )
  1h( )
  1h( )
  41h( this listing consists of output from)
  52h( the charles stark draper laboratory's jovial j3)
  34h( structured design diagrammer.)
  1h( )
  1h( )
  1h( )
  1h( )
  42h( principal designers and implementors )
  1h( )
  37h( gary w. goddard, csdl staff)
```

39h(mark h. whitworth, csdl staff)
52h(eric f. strovink, graduate student, m.i.t.)
25h(computer science division)
57h(the charles stark draper laboratory, inc., cambridge, ma.)
lh()
end
item title'no i 36 s p 17 \$
end
term \$